Process Querying: Enabling Business Intelligence through Query-Based Process Analytics

Artem Polyvyanyy[a,*], Chun Ouyang[a], Alistair Barros[a], Wil M. P. van der Aalst[b,a]

[a]*Queensland University of Technology, Brisbane, Australia*
[b]*Eindhoven University of Technology, Eindhoven, The Netherlands*

Abstract

The volume of process-related data is growing rapidly: more and more business operations are being supported and monitored by information systems. Industry 4.0 and the corresponding industrial Internet of Things are about to generate new waves of process-related data, next to the abundance of event data already present in enterprise systems. However, organizations often fail to convert such data into strategic and tactical intelligence. This is due to the lack of dedicated technologies that are tailored to effectively manage the information on processes encoded in process models and process execution records. Process-related information is a core organizational asset which requires dedicated analytics to unlock its full potential. This paper proposes a framework for devising process querying methods, i.e., techniques for the (automated) management of repositories of designed and executed processes, as well as models that describe relationships between processes. The framework is composed of generic components that can be configured to create a range of process querying methods. The motivation for the framework stems from use cases in the field of Business Process Management. The design of the framework is informed by and validated via a systematic literature review. The framework structures the state of the art and points to gaps in existing research. Process querying methods need to address these gaps to better support strategic decision-making and provide the next generation of Business Intelligence platforms.

Keywords: Process querying, process management, process analytics, process intelligence, process science, business intelligence

1. Introduction

Business Process Management (BPM) is the discipline that combines approaches for the design, execution, control, measurement, and optimization of business processes. Most of the larger organizations adopted BPM principles (e.g., designing processes explicitly). A growing, but still limited, number of organizations uses explicit BPM systems, i.e., information systems directly driven and controlled by explicit process models. Business Intelligence (BI) systems focus on the dissemination of business-related data without considering process models. Hence, one can easily witness the gap between data-driven BI approaches and process-centric BPM approaches. Process mining approaches aim to bridge this gap [1]. Like other BPM approaches, process mining is process-centric. However, unlike most BPM approaches, it is driven by factual event data rather than hand-made models.

Process mining is closely related to the term *process analytics* [2, 3] which refers to approaches, techniques, and tools to provide process participants, decision makers, and other stakeholders with insights about the efficiency and effectiveness of operational processes. The search, correlation, aggregation, analysis and visualization of process events can support insights and improvements in performance, quality, compliance, forecasting and planning, of processes operating in dynamic commercial settings. Most of the commercial tools e.g., Splunk, SAP Business Process Improvement, Pentaho, and Adobe Analytics, focus on purely structural associations in organizational information, where process execution is measured via coarse-grained events (e.g., start and end of process execution) in line with classical performance-oriented business intelligence analysis of organizational units, resources, products, services, etc. This is in stark contrast with process mining approaches that provide fact-based insights to support *process* improvements [1]. Process discovery techniques can be used to learn process models from event logs. However, process mining extends far beyond process discovery and includes topics like conformance checking, bottlenecks analysis, decision mining, organizational mining, predictive process analytics, etc. All of these process mining approaches have in common that they seek the confrontation between event data (i.e., observed behavior) and process models (hand-made or discovered automatically).

Through these developments, at least three broad contexts for process analytics can be identified to frame further development of supportive techniques. Firstly, temporal contexts are important where past and present process data are retrieved and the future behavior of processes can be projected. Secondly, process behavior needs to be understood in different organizational contexts, not only the operational level, but also at strategic and tactical levels, given reflections of processes in higher-level architecture models. Thirdly, productivity contexts nowadays focus not only on transactional considerations, through policy and performance compliance checks, but also on transformational opportunities, whereby insights into how processes can be standardized, reused, and rapidly adapted, are crucial.

Process querying studies (automated) methods for managing, e.g., filtering or manipulating, repositories of models that describe observed and/or envisioned processes, and relationships between the processes. A process querying method is a technique that, given a process repository and a process query,

[*] Corresponding author

Email addresses: artem.polyvyanyy@qut.edu.au (Artem Polyvyanyy), c.ouyang@qut.edu.au (Chun Ouyang), alistair.barros@qut.edu.au (Alistair Barros), w.m.p.v.d.aalst@tue.nl (Wil M. P. van der Aalst)

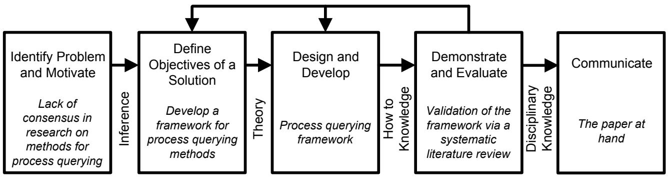
Identify Problem and Motivate
Lack of consensus in research on methods for process querying
Inference
Define Objectives of a Solution
Develop a framework for process querying methods
Theory
Design and Develop
Process querying framework
How to Knowledge
Demonstrate and Evaluate
Validation of the framework via a systematic literature review
Disciplinary Knowledge
Communicate
The paper at hand

Figure 1. DSRM process for the process querying framework.

systematically implements the query in the repository, where a *process query* is a (formal) instruction to manage a process repository. The paper addresses major limitations of techniques for process querying, which often analyze business processes on a single model scope and ignore process semantics aspects. Note that a recent survey demonstrates the lack of, and the need for, dedicated precise process querying methods grounded in execution semantics rather than the structure of business process models [4].

Concretely, this paper proposes the *Process Querying Framework*, which aims to guide development of process querying methods. Given a process repository and a process query that specifies a formal instruction to manage the given repository, the corresponding *process querying problem* consists of implementing the instruction on the repository. The proposed framework is an abstract system in which components providing generic functionality can be selectively replaced resulting in a new process querying method. The framework emphasizes unified process querying based on searching process structure and behavior, which includes the designed and observed behavior. Processes often exhibit complex alignments with higher manifestations of processes through strategic and tactical models in the organizational pyramid. Moreover, results of process querying methods must be effectively interpreted by stakeholders. The proposed in this paper framework addresses these concerns.

To develop the framework, we use (an adapted version of) the Design Science Research Methodology (DSRM) by Peffers et al [5] that follows the guidelines by Hevner et al [6] for the required elements of design research. The framework is a viable artifact that is produced as an outcome of this design endeavor (Guideline 1: Design of an Artifact, refer to [6] for details). The corresponding DSRM process is depicted in Figure 1. The process is initiated by the problem of lack of consensus in designs of methods for process querying. First, we perform a systematic CRUD (Create, Read, Update, and Delete) analysis over process repositories to identify a list of use cases for managing process repositories (Guideline 5: Research Rigor). The obtained use cases justify relevance of the problem (Guideline 2: Problem Relevance). The core objective of this work is the development of a framework for devising process querying methods. Hence, in the second step, we employ the identified use cases and CRUD operations to elicit requirements and categories of process querying problems, which in this paper are referred to as *process query intents*. Third, based on the deduced requirements, we rigorously define the *process querying problem* and *process querying method*, and use these notions as the basis for the design of the framework (Guideline 5: Research Rigor). Fourth, we validate the proposed framework via a systematic literature review (Guideline 3: Design Evaluation; Guideline 5: Research Rigor). The insights gained from this evaluation are used to iterate the design of the framework to cater for the features of the state of the art techniques for managing process repositories while still satisfying the requirements deduced from the use cases (Guideline 6: Design as a Search Process). The conducted systematic review demonstrates that the developed framework is consistent with process querying methods in prior literature. Finally, we document and discuss all the steps taken to design the framework, which is also the main contribution of this work (Guideline 4: Research Contributions), in the paper at hand (Guideline 7: Communication of Research).

The remainder of the paper is organized as follows. The next section discusses how processes manifest horizontally within the Business Process Management (BPM) lifecycle and vertically at different levels of abstraction of the organizational pyramid, and looks at use cases for managing process repositories. Based on the insights gained in Section 2, Section 3 gives rigorous definitions of the process querying problem and process querying method. Section 4 discusses the design of the process querying framework, which is based on the formal notions proposed in Section 3. Then, Section 5 suggests how the proposed framework can be positioned in light of the broader process analytics and BI. Section 6 validates the design of the framework via an extensive literature review and states the research gaps. Section 7 concludes the paper.

2. Process Querying Requirements

This section provides an exposition of process querying requirements, to develop the process querying framework, which is proposed in Sections 3 and 4. Section 2.1 provides a contextual background of BPM generally used to understand different forms of processes, how they relate to each other, and how they are managed in the BPM lifecycle. The functional requirements for process querying are then posited along the fundamental operations relevant to data querying, i.e., Create, Read, Update, and Delete (CRUD), applied to processes managed in process repositories (Section 2.2). Non-functional requirements for high performance query execution are also discussed (Section 2.3). The process of requirements elicitation is based on considering how these operations support the needs of process management as understood through relevant BPM use cases, as profiled in a comprehensive BPM survey [7]. The requirements, focused on CRUD operations, are referred to as process query intents, each bearing specific insights, per process create, read, update, and delete, which need to be supported through the proposed process querying framework.

2.1. Contexts for Process Management

We begin by considering the contextual insights in which processes are managed, and, thus, where process querying is applicable. From a broad, organizational perspective, various parts of systems, including business processes, can be seen at different levels of business to IT systems architecture, typically depicted as a pyramid [8]. Seen from this perspective,

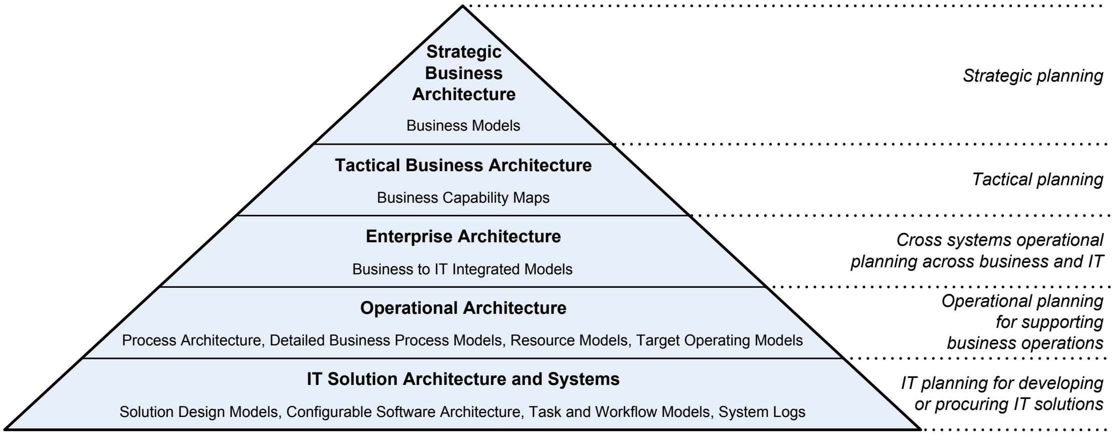
Strategic Business Architecture
Business Models
Tactical Business Architecture
Business Capability Maps
Enterprise Architecture
Business to IT Integrated Models
Operational Architecture
Process Architecture, Detailed Business Process Models, Resource Models, Target Operating Models
IT Solution Architecture and Systems
Solution Design Models, Configurable Software Architecture, Task and Workflow Models, System Logs
Strategic planning
Tactical planning
Cross systems operational planning across business and IT
Operational planning for supporting business operations
IT planning for developing or procuring IT solutions

Figure 2. Processes at different levels of the organizational pyramid.

a given process does not exist in isolation, but manifests in variety of forms and in different systems, at strategic, tactical and operational levels of an organization. Processes may be captured through dedicated modeling languages and techniques and managed through BPM systems, e.g., Petri nets, BPMN, UML Activity Diagrams, or they may be represented in other forms, e.g., task lists in task management systems and transaction processes in enterprise systems. Alternatively, they are less explicit at higher levels of the pyramid. Instead, at these levels, processes are instrumental to other methods or representations, used for broader considerations of systems planning and coordination. Figure 2 shows an organizational pyramid, illustrating a useful, *structural* context for process management—stretching from business strategy down to IT systems.

The highest-level notion of processes plays a vital role in strategic planning and the high-level representation of organizations, through business models [9]. Represented typically through strategic value chains (general activity dependencies with no control flow), processes combine with policies, target customers, product and service offerings, organizational structures and partners, to detail business models. Strategic value chains reflect not so much process flows but value accretion, together with key interactions with organizational and partner roles. When linked to processes at lower levels, they allow lower levels of processes in business and IT systems to be steered through policies and other strategic considerations of organizations.

Over the years, enterprise architecture has become an important bridge between tactical and operational levels, because it allows further details of systems (e.g., services, processes, and applications) to be aligned, thus supporting systems planning and governance from a cross-systems, i.e., enterprise purview. Enterprise architecture frameworks such as the Zachman Framework [10], TOGAF [11], and RM-ODP [12] integrate a number of modeling techniques and languages in support of this, with processes playing a central role in yielding architecture coherence. For example, in Archimate [13] used in TOGAF, processes are defined across business, application and IT infrastructure layers, and are inter-linked across these while also anchoring into other aspects such as services, resources and information. At the operational architecture level, process models take on a normative role, as opposed to being descriptive at higher levels and executable through IT systems (to use the broad positioning of processes from [7]). They guide the operations of specific business areas and are developed through individual projects. Models are captured through multi-level process architecture (from operational value chains to detailed processes) entailing many-to-many relationships between elements across levels and, thus, complex alignment challenges [14, 15].

At the lowest level, processes are a core part of IT systems design and implementations. This involves configurable, solution design models, executable models, and software applications with coded processes. Executable processes are also in the form of process or document workflows, tasks lists and other forms supported by BPM systems such as workflow management systems and task managers. Software design models also rely on process concepts to capture and configure software component dependencies, e.g., ERP solution maps and software component interactions (see exemplar software architecture of SAPs Business ByDesign [16]). Ultimately, process instances are recorded as event sequences in logs. Events capture timestamped data about executed activities and event traces are aligned to process conceptions of software interactions, e.g., transactions steps of asynchronously running business objects in ERP systems [17].

As we can see, processes are effectively refined across the architecture levels even if they are captured through different techniques and languages having either no, partial, or precise, semantics; correspondingly they are *informal* (high-level descriptive processes), *semi-formal* (lower level descriptive processes) or *formal* (normative and executable processes). Ideally, they should be aligned with processes at across all levels, therefore, requiring *correlation* of processes through query languages (akin to data correlation support in database query languages, e.g., SQL joins and correlated sub-queries).

Complementary to this *structural* context of BPM, is a *functional* context seen through the classical BPM lifecycle [7], with its comparatively narrower focus: process (re)design, implementation/configuration, and execution/adaptation. The focus of the BPM lifecycle tends to be on lower levels of architecture involving processes managed through BPM systems. Models may be (re)designed to capture requirements, refined and configured as executable models for orchestration through IT systems or as implementation logic in software code. In the execution/adaptation phase, processes are orchestrated using execution systems and event logs are generated. Through runtime execution and analysis of event data, processes may be adapted for "in-situ" improvements and overcoming errors. The execution/adaptation phase feeds back into the (re)design phase, whereby event data analysis is used to create long lasting design improvements of process models. Thus, the BPM lifecycle provides a broader context for process querying requirements, with various steps in the lifecycle offering indispensable insights for how various process create, read, update, and delete operations are combined in support of complex process management tasks.

2.2. Functional Requirements of Process Querying

To elicit requirements for different classes of management methods over processes, in this section we perform the CRUD analysis over an artifact of a process repository; note that the notion of a process repository is formalized in Section 3. The need for different CRUD operations over process repositories is justified by mapping them onto (a subset of) the comprehensive set of BPM use cases described in [7]. These use cases refer to the creation of process models and data, and their usage to improve, enact, and manage processes. The BPM uses cases were obtained by identifying interactions between artifacts such as descriptive, normative, configurable, and executable models, IT systems, event data, and a range of analysis results. Almost 300 BPM papers were mapped onto these use cases to justify their importance. The twenty use cases reported in [7] are not intended to be definitive or complete. Nevertheless, they help to structure the possible CRUD operations over process repositories. In the context of process querying, we refer to these operations as *process query intents*, which can be seen as semantic classes of management instructions for process repositories. We see process query intents as one of the configuration points of the devised process querying framework, refer to Section 4 for further details.

Design Model. Process models are produced in a number of ways including creating models for the first time via design, selection of existing model, and reuse [18] of existing models either via model merge or model composition. The Design Model use case concerns the creation of models from "scratch" by humans, capturing a current-state (as-is) or future-state (to-be) of processes. To support this, a process query intent, *Create Process*, should allow the insertion of newly designed models in a process repository, e.g., when a model being captured is saved in a process modeling tool. In addition, the intent, *Update Process*, should support updates of model designs, where the model being updated already exists in the repository. Similarly, the *Delete Process*, should support in-situ deletions of models during design, whereby the entire model is selected through the query conditions/parameters. Note, the distinction between deleting an entire model and only deleting parts of a model, where the latter can be rendered through an update query.

Select Model. The Select Model use case relates to the retrieval of process models from a repository based on structural or behavioral match of processes. Correspondingly, *Read Process* should select process models satisfying structural (graph structures) and behavioral (activity traces) based conditions. Although this use case concerns models, we extend *Read Process* to cover process models, process fragments, process instances, e.g., sequences of events in logs, and individual events. In terms of the behavior, the ability to select event traces should cover executed process, simulated behavior (based on selected workload and resource configurations) and permissible behavior (modeled but not yet executed). For example, it should be possible to provide as input behavioral activity traces and retrieve both models (based on permissible behavior) and instances (executed behavior) from a repository describing the given traces.

Specific details of processes may be projected in query results, e.g., query results may need to only include start and end activities of matched processes. Given that processes rarely exist in isolation but are linked to other processes, e.g., through use cases, collaborative process, and processes at different levels of abstraction in the systems architecture, the *Read Process* intent should support process correlations in queries. An example is to find all executable processes linked to a particular part of an operational value chain, e.g., a stage of a value chain. In terms of systems architecture, this would be expanded to finding all executable processes linked to operational processes which are linked to the corresponding part of the value chain. Complementary to exact matching, similarity match [19] is also critical for various management goals of processes, e.g., finding similarity of a set of processes to a given process or finding processes that are similar. It should be possible to reference similarity search functions as part of *Read Process* both prior to, and after search filters are evaluated (much like aggregate functions apply in SQL statements).

Models merging and model composition use cases are variants of model productions from existing models, where the resulting models rely on selection of processes from several models.

Merge Models. The Merge Models use case involves the creation of a new model based on combining parts of different models. Examples include extending a process model with parts of other models or taking different models and merging them into one model. This use case is based on an elementary step of merging through automated techniques [20], as opposed to the preparatory and intermediate steps of identifying models, updating them for fitness, etc., which involve update or deletion of parts of models. Thus, the Merge Model use case, supported through a specialized form of the *Create Process*, should take as input a set of models, allow a merge function to be used on these models, and insert the resultant model into a repository. This intent could be practically implemented in a modeling tool as part of a merge utility, where the merge function automatically generates a create process, which can then be updated and saved (committed) by a user. Since correctness of the resulting model is not guaranteed through merging, the *Update Process* applies, to support subsequent refinements of the merged model, e.g., removal or updated connection of activities to ensure correct execution.

Compose Model. The Compose Model use case involves the creation of a new model based on different, and, typically, reusable models. Like merge models, a specific variant of *Create Process*, applicable for composition, should take as input a set of models, allow an algorithmic composition of these models, and insert the composition into the repository. Unlike, merge model, the composed models parts can be related to the original models, and a corresponding correlation should be explicitly captured. Given the more structured nature of composition, the subsequent refinement of models for correctness through the *Update Process* queries is less likely to be required.

Following on from use cases concerning model production, we now consider use cases related to process execution. These involve the creation of executable models, typically from non-executable models, e.g., normative process models designed at the operational architecture level, the execution of models, the creation of process instance through events, process monitoring and runtime process adaptation.

Refine Model. The need for the detailing of a higher-level model into an executable model, through the Refine Model use case, can, in fact, be generalized for model refinement across different levels of systems architecture, refer to Figure 2. Each level uses modeling techniques and languages with different degrees of semantics, with executable models needing to be precise and free of errors so that they can be executed. Moreover, platform-specific technical configuration details need to be present, e.g., message correlations, data object mapping to required schemas, for execution readiness. As such, *Create Process* should support the initial creation (including versioning) of a refined model and linking it and specific parts of refinement with the parent model(s). To support this use case, the *Update Process* applies for interim saves during refinement steps and flagging models that they are in a verified, error-free form for execution.

Enact Model. The Enact Model use case, relating to the interpretation of executable models by BPM systems, has somewhat a subtle application for process querying. Model execution centers on the selection, scheduling, and execution of activities, through execution engines. While the core execution components control the reading, scheduling and internal state management of individual activities, the goals of efficient memory management and reduced latency require that parts of models pre-fetched into memory, ahead of execution, akin to database query execution strategies. For this, *Read Process* should be used to support sequential "pre-fetch" of process models in fragments aligned with platform specific constraints, e.g., memory blocks. Thus, we envisage process querying to be better exploited by process execution engines, down to low-level technical concerns, i.e., read queries are generated through execution components of BPM systems. Note, given that different parts of processes are possible for downstream execution through choice constructs in models, activity traces based on model structure and behavior could be used as part of pre-fetch optimization strategies, in line with database query execution optimization strategies [21].

Log Event Data. When processes are executed the instances that are generated and run are recorded as events in system logs, addressed through the Log Event Data use case. This can be supported through *Create Process* and *Update Process* query intents, which should specify instructions to create event logs, traces, and events, as well as update traces by inserting fresh events into traces in tandem with process execution. Timestamps and other logistical details provided by execution engines can be logged in events.

Monitor. The Monitor use case, relating to the active reading of system logs at runtime for evaluating responsiveness, throughput, resource cost, and other performance indicators, can be supported via *Read Process* queries, applied over different process instance sets: individual, case based, systems wide, etc.

Adapt While Running. The Adapt While Running use case addresses ad-hoc, or permanent, model changes required during runtime due to emergent requirements, e.g., making sequentially ordered activities run in parallel in urgent situations. These practices can be supported through *Update Process* queries. Note that the impact of runtime adaptations is on process instances, as newly generated events will result from changed models, which are inter-linked to events of the previous models. Thus, updated models need to be versioned within a current change release cycle, as well as against all the previous changes. A specific challenge is to ensure that updates are made with the proper integrity, and models used as input for change are checked and compared against the intended (to-be) processes to ensure proper continuity of execution, e.g., stopping a process in the middle of iterated activities (within a loop) could result in integrity issues.

In addition to being designed, configured, implemented, and executed, processes can be used for non-trivial analysis related to performance and verification. Of these, we consider the former for query requirements.

Analyze Performance Based on Model. The Analyze Performance Based on Model use case concerns the simulation of executable process models for performance analysis in terms of response times, latencies, resource utilization, throughput, etc. Simulation techniques focus on specialized analysis such as queueing networks or Markov chains to compute the expected performance. The results of model simulation, resulting in simulated process instances, can be stored in repositories through the use of *Create Process* and *Update Process* queries, generated by simulation tools. In addition, *Read Process* queries should be used through these tools to retrieve simulated processes for comparing how different systems configurations impact the performance of models. *Read Process* should also support diagnosis of individual processes and the aggregate analysis of sets of process instances, e.g., process analytics functions.

Further use cases concerning process analysis involve both process models and event data, concerning conformance checking and performance analysis.

Check Conformance Using Event Data. The Check Conformance Using Event Data use case covers design-time, runtime, and post runtime analysis to check that processes comply with business rules, business requirements and model specifications. One consideration is conformance checking of normative or executable processes against high-level, descriptive models. Furthermore, if the models being compared are formal, behavioral conformance can be checked based on permissible sequences of activities. Where processes have been executed, conformance of the specified behavior in models can be checked against observed behavior in event logs, to ensure that the right sequences of activities have been executed. Accordingly, *Read Process* queries should retrieve both models and process instances for structural, behavioral and specified vs. observed behavioral conformance checking. Queries reflecting this intent should be integral to higher-level process analytics, since this is a key enabler for process conformance checking, e.g., conformance checking for business areas and the generation of reporting capturing statistical analysis for business compliance thresholds. A major consideration for conformance checking lies in collaborative processes, where cross-process interactions need to be checked for correct executions and conformance to business rules, e.g., related to service contracts.

Analyze Performance Using Event Data. The Analyze Performance Using Event Data use case covers runtime monitoring and post runtime analysis to check processes for execution characteristics such as response times, latencies, and throughput of processes. The intent of *Read Process* is, thus, similar to that one for the Check Conformance Using Event Data use case, but applied for a different set of analysis considerations. Individual and aggregated event data should be selected and compared against key performance indicators. Future performance could also be checked through performance profiles of activities determined from historical data and used as input for simulation or prediction techniques. Thus, this intent should be integrated into process analytics techniques concerned with performance.

The final set of BPM use cases we consider relate to the use of diagnostic information and event data that can be used to repair, extend, or improve models.

Repair Model. The Repair Model use case refers to the problem of transforming a given process model to better reflect the observed deviating behavior recorded in the corresponding event log [22, 23]. The challenge is to obtain a good quality model, e.g., structurally simple, which reflects all the important observed behavior as closely as possible. The Repair Model use case can be supported using *Update Process* and *Delete Process* intents over process repositories by providing fine granular operations for transforming models in the repository to describe, or not describe, certain processes that are recorded or, respectively, not recorded in the corresponding event logs.

Extend Model. The Extend Model use case relates to the use of event data in process models to extend the description of process models. Event data may be annotated with additional information such as the timestamp of the event, person/resource executing or initiating the activity, and data elements recorded with the event. This information can be used to enrich the process model and extend it beyond a control-flow oriented model with additional perspectives. For example, timestamps of events may be used to add delay distributions to the model. Data elements, used as part of decisions, can be used to enrich models with business rules. Resource information can be used to confer role assignment on activities in the model. To support this use case, *Read Process* should be able to retrieve and correlate process model and event data at the level of activities. Once users have viewed, analyzed and saved the extended models, *Update Process* should allow extended models to be stored in a repository. Alternatively, a *Create Process* should support the creation of extended models, as new model versions.

Improve Model. The Improve Model use case focuses on how performance related diagnostics can be used to generate alternative process designs which represent improvements of processes, e.g., optimization of resources or reducing response times. Process models combine with event data for inferring alternative behavioral sequences and activity configurations for processes. Thus, *Read Process* should retrieve models and event data, which are used as input into process analysis targeting improvements. As part of this, *Read Process* should be able to infer gaps between expected and observed behavior in order to identify improvements concerned with addressing gaps.

2.3. Non-functional Requirements of Process Querying

The aforementioned intents involving CRUD operations on process repositories impose high performance and scalability requirements for process querying engines, considering the high computation overheads involved in matching processes, active monitoring, and complex conformance checking. Mature strategies and techniques are available through contributions to database querying over many years. These have had to content with core challenges of query processing, which are also relevant to processes. We identify salient aspects drawn from this tradition for key non-functional considerations of process querying.

Scanning of logical storage resources, e.g., databases tables, for processes and process models, inclusive of structural and behavioral data, and covering both executed and simulated data, needs be done in real-time and scaled out through parallel processing. While scanning is efficient through in-memory database technology, the explosive growth of "big data" through event streaming applications, e.g., sensor data streams in Internet of Things applications, nonetheless warrants the use of indexing, e.g., B-Tree or R-Tree, to be created for process data, as alternative access paths to sequential process repository scans. Proposals for dedicated *indexing* strategies for processes involve indexing activity traces and subgraph isomorphism algorithms [24, 25]. We include under the banner of "indexing" pragmatic considerations of pre-processing data, e.g., prejoining, to reduce computational latency and support rapid retrieval.

Prior to query execution, it is crucial that data distributions of processes and indexes be profiled, since ranges of attribute values, sizes of processes and many other insights can be used to eliminate more expensive access strategies and choose the most optimal ones, e.g., sequential scan or specific index. As with database query engines, *process statistics* should be collected periodically on process repositories. *Query optimization* addresses execution efficiency and is particularly relevant for read, update, and delete queries, which require evaluation of complex search conditions. It involves the transformation of a query into an equivalent form (*logical query optimization*), as well as the selection of optimal access paths and generation of access plan (*physical query optimization*).

In line with contemporary search engines, frequently executed queries and corresponding results should be "reusable", subject to the consistency of data, related to process update frequency. Static data, through process execution history, or static reference models are good candidates for efficient retrieval strategies through *caching*. Instrumental for caching decisions is the availability of *process querying statistics*.

3. Process Querying

Based on the various requirements reported in Section 2, this section gives rigorous definitions of the *process querying problem* and *process querying method*. The process querying problem is proposed as an overarching problem of managing, e.g., *filtering* or *manipulating*, (repositories of) models of observed or envisioned processes, and models that describe relationships between processes. The proposed formal notions set the basis for the design of the framework presented in the next section.

Processes are properties of dynamic systems, where a *dynamic system* is a system that changes over time, e.g., a process-aware information system or software system. A *process* is an ordering of events that together strive to achieve a goal state. A *state* is a characteristic of a condition of the system at some point in time, i.e., all the information that is relevant to the system at a certain moment in time. An *event* indicates an instantaneous change of the current state of the system, e.g., a change of the traffic light or change of the reading on a digital clock. One can distinguish an event from other events via its attributes and attribute values, e.g., a timestamp 'Mon 01-11-2016 11:06'. Events are often induced by *activities* performed by the system. An event may, for instance, indicate a start or a completion of the activity. Examples of activities include reading this article, manufacturing a product, or handling a bank transfer.

Let $\mathcal{U}_{an}$ be the set of all *attribute names*. We introduce three special attribute names $time, act, rel \in \mathcal{U}_{an}$, where *time*, *act*, and *rel* are the 'timestamp', 'activity', and 'relationship' attribute names, respectively.

Let $\mathcal{U}_{av}$ be the set of all *attribute values*.

Definition 3.1 (Events). An *event* e is a partial function from attribute names to values, $e : \mathcal{U}_{an} \nrightarrow \mathcal{U}_{av}$. ⌟

By $\mathcal{E}$, we denote the set of all events. We specify three special classes of events: $\mathcal{E}_{time}$, $\mathcal{E}_{act}$, and $\mathcal{E}_{rel}$. By $\mathcal{E}_{time}$ and $\mathcal{E}_{act}$, we denote the sets of all events with timestamps, i.e., $\mathcal{E}_{time} := \{e \in \mathcal{E} \mid time \in dom(e)\}$, and activity names, i.e., $\mathcal{E}_{act} := \{e \in \mathcal{E} \mid act \in dom(e)\}$, respectively. To capture the vertical process relationships in the context of the organizational pyramid, refer to Section 2.1, we introduce special events $\mathcal{E}_{rel}$. Let $\mathcal{U}_{rel} := \wp(\mathcal{E}) \times \wp(\mathcal{E})$, where $\mathcal{U}_{rel} \subset \mathcal{U}_{av}$. Then, $\mathcal{E}_{rel}$ is the set of all events with the relationship attribute whose values are in $\mathcal{U}_{rel}$, i.e., $\mathcal{E}_{rel} := \{e \in \mathcal{E} \mid rel \in dom(e) \wedge e(rel) \in \mathcal{U}_{rel}\}$. Hence, an event in $\mathcal{E}_{rel}$ refers to two sets of events that may stem from two different levels in the organizational pyramid.

A *process* describes that some events are ordered, i.e., some events precede other events, while some events are unordered, i.e., for some events in the process the precedence relation is unknown.

Definition 3.2 (Processes). A *process* is a partially ordered set $\pi := (E, \leq)$, where $E \subseteq \mathcal{E}$ is a set of events and $\leq \subseteq E \times E$ is a partial order over E, i.e., $\leq$ is a reflexive, antisymmetric, and transitive binary relation. ⌟

Two unordered events of the process are often interpreted as such that may be enabled simultaneously or can occur concurrently, refer to [26–28]. A process that is a totally ordered set is a *trace*.

A behavior is a collection of processes that can contain identical processes.

Definition 3.3 (Behaviors). A *behavior* b is a multiset of processes. ⌟

By $\mathcal{B}$, we denote the set of all behaviors. Intuitively, multiple instances of a process in a behavior represent the fact that the process was observed multiple times in the real-world.

One can describe behaviors using conceptual models. A conceptual model is composed of the explicit model and the implicit model [29]. The *explicit model* is the set of all statements explicitly made using some modeling language in order to construct the model. The *implicit model* is the set of all statements that can be derived from the explicit model using deduction rules of the modeling language. We refer to a conceptual model that describes behaviors as a behavior model. Based on the requirements listed in Section 2, we introduce four classes of behavior models. A *behavior model* is a (formal) description of a collection of real-world or envisioned processes of the same nature (i.e., processes that aim to achieve the same goal), which serves a particular purpose for a target audience. Behavior models of real-world processes encode processes that follow well-known (already experienced) sequences of events. Alternatively, behavior models of envisioned processes describe never observed sequences of events. For example, event logs [1] are formal models of events generated by executions of process-aware information systems and, hence, are examples of behavior models that capture real-world processes. A design of a new computer algorithm [30] or a specification of an innovative business process [31] are examples of models of envisioned processes.

Let $\mathcal{A} \subset \mathcal{U}_{av}$ be the set of all *activities*. Let $\mathcal{U}_{ms}$ be the set of all (explicit) *model statements*. Then, $\mathcal{M} := \wp(\mathcal{A}) \times \wp(\mathcal{U}_{ms})$ is the set of all *activity models*, where $\circledcirc := (\emptyset, \emptyset)$, $\circledcirc \in \mathcal{M}$, is the special 'empty' model. Thus, an activity model $M := (A, S)$ consists of statements S over activities A.

Definition 3.4 (Behavior models). A *behavior model* is a pair (M, B), where $M \in \mathcal{M}$ is an activity model and $B \subseteq \mathcal{B}$ is a set of behaviors. We introduce four types of behavior models:

- An *event log* is a behavior model $(\circledcirc, \{b\})$, where $b \in \mathcal{B}$ is a finite multiset of finite traces over $\mathcal{E}_{act}$.
- A *simulation model* is a behavior model $(M, \{b\})$, where $M = (A, S) \in \mathcal{M}$ is a nonempty model and $b \in \mathcal{B}$ is a finite multiset of finite processes over $\mathcal{E}_{act}$ such that for every event e in a process in b it holds that $e(act) \in A$.
- A *process model* is a behavior model (M, B), where $M = (A, S) \in \mathcal{M}$ is a nonempty model and every $b \in B$ is a set of processes over $\mathcal{E}_{act} \setminus \mathcal{E}_{time}$ such that for every event e in a process in $b \in B$ it holds that $e(act) \in A$.
- A *correlation model* is a behavior model (M, B), where every $b \in B$ is a multiset of processes over $\mathcal{E}_{rel}$ such that if $M = (A, S)$ is a nonempty model in $\mathcal{M}$, then for every event e in a process in $b \in B$ it holds that $act \in dom(e)$ and $e(act) \in A$. ⌟

We say that M and B are, respectively, the *explicit component* and the *implicit component* of the behavior model (M, B). A behavior model $(M, \varnothing)$ is *informal*; no implicit statement can be deduced from M. A behavior model $(M, \{b\})$, where $b \in \mathcal{B}$, is *formal*; the explicit component of a formal behavior model induces one behavior, i.e., all the implicit statements are deduced from M deterministically. A behavior model (M, B), $|B| > 1$, is *semi-formal*; the explicit component of a semi-formal behavior model can be interpreted as one of the behaviors in B reflecting that the deduction rules of the modeling language used to construct the explicit model are nondeterministic. Behavior models are immense information resources. A behavior model characterizes a dynamic system by describing often an infinite collection of processes that suggest the ways to lead the system to a potentially infinite number of states [32].

Event logs are studied within the process mining discipline [1]. Event logs are composed of traces, where each trace is a finite sequence of events that denotes the finite "life" of a process observed and recorded in the real-world. The core competence of process mining is to connect the observed behavior, i.e., an event log, to an activity model, e.g., a Petri net or BPMN model, which induces the implicit model that resembles the event log as closely as possible. The multiplicity of a trace in the only behavior of the implicit component of the event log denotes the number of times the trace was observed. Because the original link between the traces and the corresponding activity model is absent, the explicit component of the event log is the empty model. However, events in traces of event logs have the *act* attribute to refer to activities that induced them.

A *simulation model* is an activity model together with a finite imitation of its operations in the real-world [33]. The activity model represents the key behaviors of the system, while the imitation is derived based on the deduction rules of the modeling language used to construct the activity model. Thus, every event of the implicit component of the simulation model has the *act* attribute to refer to the activity that induced the event. Note that the only behavior of a simulation model (or that one of an event log) encodes only a part of the behavior that can be deduced from the corresponding activity model.

A *process model* is an activity model together with a set of all possible behaviors that can be deduced from the statements in the activity model [34, 35]. A behavior induced by the explicit component of a process model can be infinite, e.g., the set of all processes induced by the model which prescribes that every process must start by repeating activity X arbitrary number of times before performing activity Y and then concluding that process is infinite. In addition, a behavior induced by the explicit component of a process model can contain an infinite process, e.g., the maximal trace induced by the model that prescribes to repeatedly perform activity Z is infinite. To capture that events in the implicit components of process models are envisioned, i.e., did not occur in the real-world, they do not have timestamps. Though the implicit components can be infinite, it is expected that in most of the practical process querying scenarios explicit components of behavior models will be finite, as one must be able to store models on a computer.

A *correlation model* is a behavior model in which every event of the implicit component has the *rel* attribute to refer to two sets of related events. For example, conformance checking [36] refers to the problem that given an event log and a formal process model checks whether traces in the event log are in accordance with processes of the model. An *alignment* is an example of a correlation model [36]. An alignment between a trace of an event log and a process of a process model is a sequence of moves, where a move is a pair in which the first component refers to an event in the trace and the second component refers to an event in the process. In general, correlation models allow relating behavior models at different levels of abstraction and/or granularity, i.e., vertically and/or horizontally within the organizational pyramid, cf. Figure 2.

In this work, we restrict the scope to the four aforementioned classes of behavior models. However, we envision that future works on process querying will introduce new classes of behavior models, and will augment the proposed notion of a behavior model to cater for the emerging requirements.

A *process repository* is an organized collection of behavior models. Let $\mathcal{U}_{re}$ be the set of all *repository elements* which are not behavior models, e.g., folder structures for organizing behavior models, names and values of behavior model attributes (such as authors and versions of behavior models), etc.

Definition 3.5 (Process repositories). A *process repository* is a pair (P, R), where P is a collection of behavior models and $R \subseteq \mathcal{U}_{re}$ is a set of repository elements. ⌟

By $\mathcal{U}_{pr}$, we denote the set of all *process repositories*. Let $\mathcal{U}_{qi}$ be the set of all *query intents* that represent the abstract semantics of methods for querying over process repositories. We propose that $\mathcal{U}_{qi}$ consists of Create, Read, Update, and Delete (process) query intents, refer to Section 2.2 for details. Then, $\mathcal{U}_{pq} := \mathcal{U}_{qi} \times \mathcal{U}_{qc}^{*}$ is the set of all *process queries*, where $\mathcal{U}_{qc}$ is the set of all *query conditions*, or query parameters. A process query contains a sequence of parameters to allow distinguishing between several parameters of the same type. For example, a process query can capture an instruction to update, using the update query intent, a process in one of the behaviors of a process model with a given fresh process. Among other conditions, this query should contain two conditions: the process that needs to be updated and the fresh process. To distinguish between the two processes, they must be placed at two different positions in the sequence of conditions of the query. Note that the above described query instruction on the level of behaviors of process models implies changes in the activity models of the resulting updated process model.

Finally, a process querying method is an (automated) technique that given a process repository and a process query systematically performs the query on the given repository. A result of a process querying method is, again, a process repository which implements the query on the input repository.

Definition 3.6 (Querying methods).
A *process querying method* is a function $m : \mathcal{U}_{pr} \times \mathcal{U}_{pq} \rightarrow \mathcal{U}_{pr}$. ⌟

A combination of a process repository and process query specifies the *process querying problem*. To solve process querying problems, process querying methods rely on theoretical computer science fundamentals, e.g., results in distributed and parallel computing, model checking, and formal methods. Section 6 reports on a comprehensive review of the state of the art methods for process querying.

4. The Process Querying Framework

Based on the formal notions from the previous section, this section proposes the Process Querying Framework (PQF). Section 4.1 discusses components and logical parts of the framework. Section 4.2 discusses core design decisions that one needs to take when designing a process querying method. Section 4.3 states several challenges that may emerge as consequences of taken decisions. Finally, Section 4.4 suggests that every process querying method results from a compromise between solutions to the stated challenges.

4.1. Components and Parts of the Framework

The PQF is an abstract system in which components providing generic functionality can be selectively replaced resulting in a new process querying method. A schematic view of the framework using an ad-hoc notation is shown in Figure 3. In this notation, rectangles denote *active components*, i.e., actions to be performed by the process querying methods. Ovals are used to represent *passive components*, i.e., objects and aggregations of objects that are inputs and outputs of actions. To denote that a passive component serves as an input to an action, an arc is drawn to point from the component to the action. An arc that points from an action to a passive component suggests that the action produces the component as an output. Dashed lines are used to encode the aggregation relationships, where a component that is used as an input to an action contains the adjacent passive components, e.g., a process repository is an aggregation of behavior models (refer to Definitions 3.4 and 3.5). The framework is logically divided into four parts that are 'responsible' for (i) designing process repositories and process queries, (ii) preparing and (iii) executing process queries, and (iv) interpreting results of the process querying methods. In Figure 3, each part of the framework is enclosed in an area denoted by the dotted border. Next, we detail each of the four parts.

Model, Simulate, Record, and Correlate. This part of the framework (see the top of Figure 3) is responsible for acquiring/ constructing, behavior models and formalizing/designing, process queries. Behavior models can be acquired in several ways. They may stem from manual, semi-, or fully-automated exercises. Examples of automated exercises include model discovery using process mining techniques [1] and model construction using process querying, e.g., a fresh model may result from executing an update query. Behavioral models can be constructed by recording or simulating execution traces of systems, and by correlating steps of two different processes. All these alternatives are captured by the *Modeling*, *Simulating*, *Recording*, and *Correlating* active components, refer to Figure 3. Examples of behavior models include computer programs, business process specifications (e.g., EPC, BPMN, YAWL, BPEL models), formal models of computation (finite automata and Petri nets), event logs [1], and alignments [36].

Process querying demands mathematically precise and unambiguous specifications of instructions for managing process repositories. A *process querying instruction*, or a *query*, specifies how processes, behaviors, and behavior models should be manipulated in a process repository. It is composed of a *query intent* and a list of *query conditions*, refer to Section 3 for details. The intent specifies the abstract semantics of the query, e.g., to *retrieve*/*read* processes, behaviors, or models, or to *remove*/*delete* some processes in certain behavior models. Query conditions are used to parameterize intents to obtain precise querying instructions. For example, the aforementioned intents must be supplied with conditions to specify what processes to retrieve and remove. Thus, a query condition may be specified as a collection of processes, i.e., a behavior model. Note that two queries with different intents may have different query conditions. For example, an update query may use three conditions to specify in what collections of processes (i), which old processes (ii) should be replaced with which fresh processes (iii). For a delete query, it may suffice to use two conditions that tell in what collections of processes (i) which processes (ii) should be removed.

The PQF relies on formal representations of queries. The *Formalizing* component of the framework takes a process querying instruction as input and produces a (process) query that captures the instruction in a formal language. Concrete instantiations of the framework may rely on manual, semi-, or fully-automated components responsible for the formalization of process querying instructions.

Prepare. The "Prepare" part of the framework (see the bottom left of Figure 3) is responsible for making process repositories ready for efficient querying. This part includes components for constructing dedicated data structures that can speed up execution of process queries. We suggest constructing these data structures offline. They often require additional storage space to maintain the extra copy, or even several copies, of an indexed repository, behavior, or process. The framework suggests two methods for preparing for querying: indexing and caching. In databases, *indexing* is a technique to construct a data structure to efficiently retrieve data records based on some attributes. In

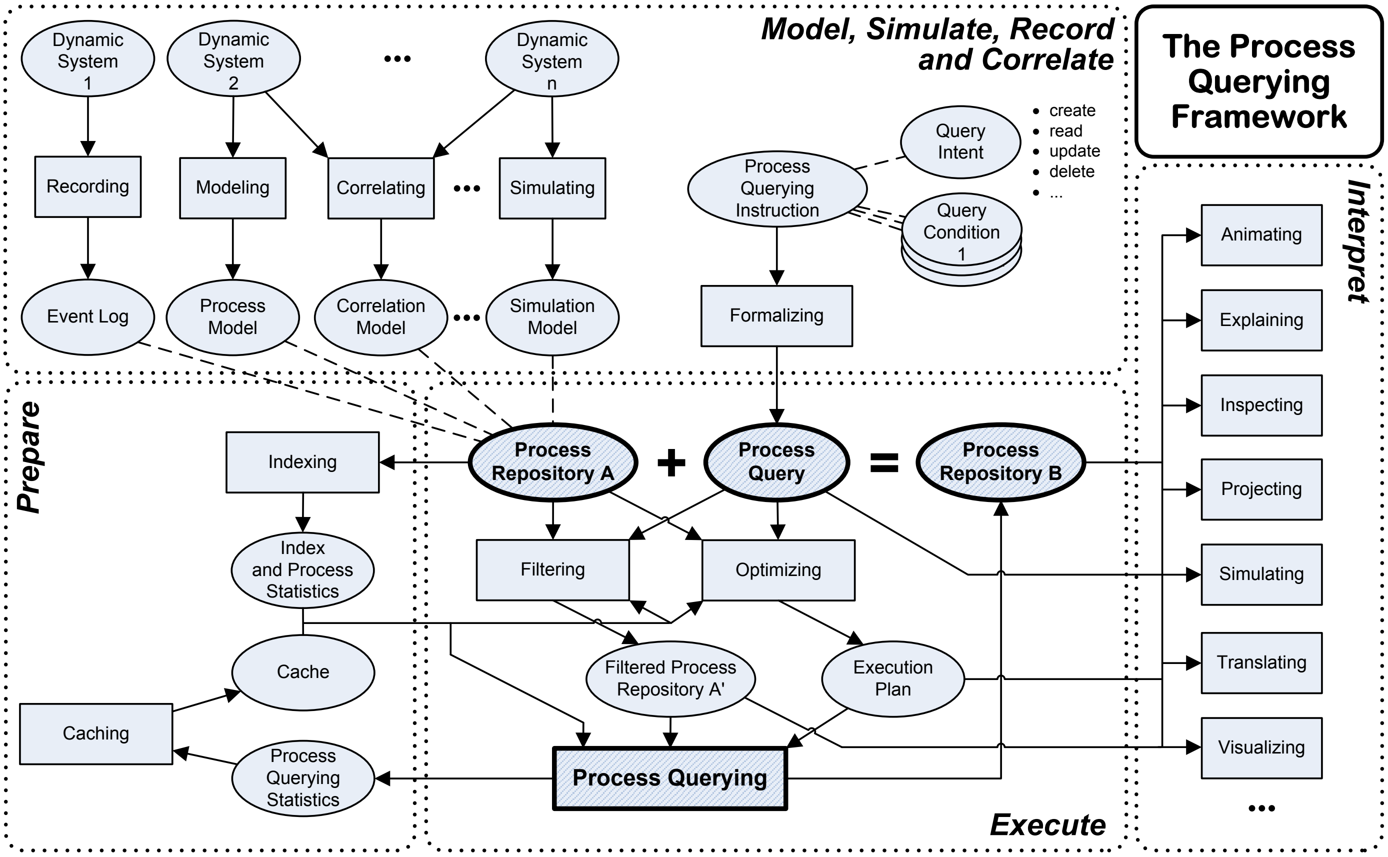

Figure 3. A schematic view of the Process Querying Framework.

computing, *caching* is a technique to store data so that future requests to that data can be served faster, where the data that gets stored in a cache might be the result of an earlier computation. Similar ideas can be used to speed up executions of process querying methods.

The *Indexing* component of the framework takes a process repository as input and constructs its alternative representations, i.e., an *index*. An index can be larger in size than the repository. By using an index, process querying methods can achieve different space-time tradeoffs, i.e., they can trade increased space of input process representations for a decrease in execution times of process queries. The *Indexing* component is also responsible for collecting statistics over properties of repositories, behaviors, processes, and index, denoted by *Process Statistics* in the figure. Process statistics can be used to 'guide' the execution of queries. For example, a process querying method can proceed by first executing queries over small models. The rationale behind this strategy is based on the assumption that execution of queries over small behavior models will take significantly less time and, hence, first query results can be obtained faster.

The *Caching* component of the framework relies on *Process Querying Statistics* to decide which (parts of) query results to store for later reuse. Process Querying Statistics may include aggregate information on the execution of process queries and evaluation of process query conditions, e.g., their frequencies. It gets regularly updated by the *Process Querying* component and, thus, is denoted as its output in Figure 3. Results of frequent process queries can be stored in a *cache* and looked up at a later stage, e.g., when the user requests to evaluate the query again. Similarly, results of evaluating frequent process query conditions can be stored and reused, even when new requests to these conditions originate from fresh queries.

One can adopt other standard optimization approaches for process querying. These include parallel computing (e.g., map-reduce), algorithm redesign (e.g., stochastic and dynamic optimization), and hardware acceleration (e.g., in-memory databases). However, these approaches are often inherent to the design of techniques that get optimized. In contrast, techniques in the "Prepare" part of the framework are complement optimizations that are 'orthogonal' to the design of the process querying methods.

Execute. The "Execute" part of the framework (see the bottom right of Figure 3) is responsible for executing process queries. Prior to executing a query, to avoid unnecessary computations, one can filter the repository to remove models, behaviors, and/or processes that are irrelevant for the purpose of the query. For example, if a query requests to retrieve models that describe a process with an event that refers to a given activity, it makes no sense to execute the query over the models that do not contain that activity.

The *Filtering* component is responsible for filtering process repositories. The component takes a *Process Repository* and *Process Query* as input and produces a *Filtered Process Repository*. The filtered repository is the input repository with some of its parts marked as irrelevant for the purpose of executing the query. To identify the irrelevant parts of a repository, the component uses information in *Index*, *Process Statistics*, and *Cache*; note that information kept in these components can be used to anticipate query results.

The *Optimizing* component is responsible for query optimization. It takes the same input as the *Filtering* component and produces an *Execution Plan*—a list of instructions that aim at executing the input query using the least possible amount of resources. The *Optimizing* component may implement two types of optimization: logical and physical. A *logical optimization* entails reformulating a given query into an equivalent but easier – which usually means a faster – to execute query. A *physical optimization* is responsible for determining efficient means for carrying out instructions in a given execution plan.

Finally, the *Process Querying* component takes *Filtered Process Repository*, *Execution Plan*, *Index*, *Process Statistics*, and *Cache* as input and applies a process querying method to produce a fresh *Process Repository* that implements the query. Based on the query and its result, the component updates *Process Querying Statistics*. The *Filtered Process Repository* and *Execution Plan* are the *critical inputs* of the *Process Querying* component. The query cannot be executed without these inputs. All the other inputs can, in principal, be empty. We refer to the resulting *Process Repository* as the *critical output* of the component.

Interpret. Process querying can lead to two outcomes: (i) the query instruction is successfully implemented in the resulting repository, or (ii) available resources are not sufficient to execute the query. The latter situation may arise when managing vast (possibly infinite) collections of processes using finite resources, e.g., finite computer memory or limited processing power. If it is impossible to execute a query due to the physical limits of available resources, one can proceed in several ways. Sometimes it may suffice to reformulate the original query to give up on the precision of the expected result. Alternatively, one may try to optimize the querying method to handle the specifics of the original query. Some approaches to managing vast collections of processes include the use of symbolic techniques (e.g., binary decision diagrams), manipulations with structural regularities in behavior models, and rigorous abstractions of processes.

Once a query gets executed, its result should be communicated to the user. Because queries can formulate elaborate instructions that induce manipulations over large data sets, the user requires support to facilitate understanding of query results. To this end, the framework includes a dedicated part that is responsible for interpreting results of executed queries (see the right of Figure 3). The common goal of all the components of the "Interpret" part of the framework is to contribute towards user's better comprehension of process querying results. They are inspired by the various means for improving comprehension of conceptual models proposed by Lindland et al [29]. The components perform whatever it takes to make the result of executing a given process query easier to understand by the user. To identify and explain the differences between the original repository and the resulting repository as well as the reasons for the differences, the components of the "Interpret" part of the framework rely on all the critical inputs and outputs of the *Process Querying* component, i.e., they take all the bits and pieces that are used to execute the query as input.

To foster understanding of process query results, one can introduce techniques for inspecting them. Indeed, the user can understand a concept or phenomenon by inspecting, or reading, it. Various approaches can be proposed to facilitate and guide the process of reading query results. The inspection can be supported by explanation notes predefined by process analysts and domain experts. Comprehension of query results can be improved by presenting them diagrammatically. The derived visual artifacts can be animated to demonstrate dynamics of processes that were effected during execution of the query. If artifacts that encode query results get large in size, their comprehension can be stimulated by projecting, i.e., hiding, some of their parts. One can use simulations to induce sample processes that were effected by the query. Finally, query results can be translated to various formalisms that are easier understood by the users.

We anticipate that new means for explaining query results will emerge, as methods for process querying will become more mature. When explaining results of queries to the users, we recommend to apply best practices in design of easy-to-comprehend process models and related artifacts, refer to [37, 38] for details.

4.2. Design Decisions

A design decision, or a design rationale, is an explicit argumentation for the reasons behind a decision made when designing a system or artifact. When designing a new process querying method, one needs to take various design decisions. Some fundamental decisions that emerge from the PQF are discussed below.

DD1: Which behavior models to support? First and foremost, one needs to decide which behavior models will be supported by the envisaged process querying method. For example, one may wish to develop a method for managing event logs. This method will most likely be different from the one that manages correlation models (even if both methods support the same query intents). The choice of behavior models implicitly restricts the class of processes, or languages (in the terminology of computation theory [39]), supported by the method. For example, if one restricts behavior models to process models captured using finite automata, the class of supported processes will be limited by the class of regular languages [39].

DD2: Which processes to support? An activity model, as defined in Section 3, can be interpreted as such that describes several different collections of processes, each induced by a different semantics criterion. For example, an activity model can be interpreted using the finite, infinite, or fair process semantics. According to the finite process semantics, an activity model captures a collection of processes that lead to a terminal state. If one considers the infinite process semantics, an activity model can describe processes that never terminate, i.e., processes that describe infinitely many events. Thus, an infinite process strives to, but never achieves, its goal state. A fair process can be finite or infinite. A process in which an event is enabled for execution over and over again but does not get performed from some state on is unfair [40]. A not unfair process, as per the above principle, is said to be strongly fair. In [41], the authors study strong fairness and several other types of fair process semantics criteria. The choice of the correspondences between activity models and collections of processes that they are associated with defines the problem space for process querying methods, i.e., it implies processes to consider when executing process queries.

DD3: Which process queries to support? When devising a process querying method, one needs to decide which types of queries the method will support. The design of a query entails choosing its *intent* and *conditions*. A choice of the intent suggests the semantics of the query, e.g., to create, read, update, or delete behavior models, or parts thereof, in a process repository. Query conditions are used to specify query parameters. The choice of supported queries determines the expressiveness of the process querying method, i.e., its ability to describe various problems for managing process repositories.

4.3. Design Challenges

Next, we discuss some challenges that one may face when developing a process querying method. These challenges stem from the aforementioned design decisions. The subsequent discussions are not meant to be exhaustive. We envision that new challenges will arise in future to address emerging requirements.

DC1: Computability. Process queries must be computable, i.e., it should be possible to solve process querying problems using algorithms (preferably on a wide range of inputs). This poses a significant design challenge. Note that there exist process querying problems that are known to be undecidable, i.e., it has been demonstrated that they cannot be solved. For example, process queries can be specified as temporal logic formulas [42]. However, certain temporal logic formulas are undecidable on some classes of processes [43]. A query that cannot be computed is of no help to the user. Thus, one needs to ensure that queries supported by the devised process querying method are decidable on the class of the supported behavior models.

DC2: Complexity/Efficiency. Process querying aims at providing valuable insights into operational processes and recorded business cases of modern organizations. In addition to the aforementioned use cases, refer to Section 2, process querying methods should support users in learning processes, behaviors, and models contained in process repositories, i.e., they should support exploratory querying [44]. This calls for techniques capable of executing queries efficiently. Hence, another fundamental challenge of designing a good process querying method is to propose a fast method that executes using small memory footprints. One can measure the efficiency of a process querying method using techniques in computational complexity theory, which study computation time and storage space required to solve a computational problem.

DC3: Expressiveness/Suitability. Process querying methods should offer a great variety of concepts and principles to capture and exercise in the context of process querying. Hence, the third fundamental challenge of process querying is to achieve full expressiveness, i.e., the ability to capture all suitable (useful to the users) process queries that specify instructions for managing process repositories. A process querying method should support all the useful (as perceived by the users) process queries. The suitability of process queries can be assessed empirically or, similar to [45], by identifying reoccurring patterns in queries.

4.4. Process Querying Compromise

A process querying method can be characterized by the expressiveness of the supported process queries that result from a compromise of selecting computable, efficient, and suitable queries. Given answers to design decisions DD1 and DD2, i.e., which behavior models and processes to support, cf. Section 4.2, one should aim at supporting as many useful process queries as possible.

Let D be the set of all computable, or decidable, process queries (in the context of the supported behavior models). Let $E \subseteq D$ be the set of all process queries that can be computed efficiently. Finally, let S be the set of all process queries that are perceived by the users as suitable, or useful. Then, queries in $E \cap S$ are the queries that should be supported by the process querying method. Ideally, it should hold that $S \subseteq E$, i.e., all the queries that are of interest to the users can be computed efficiently. However, in practice, it may be impossible to fulfill the requirement of $S \subseteq D$. Then, one should strive to improve the efficiency of methods for computing queries in $(S \cap D) \setminus E$. Note that to achieve completeness in this endeavor, one should know the decidability of every query in S.

5. Query-Based Process Analytics and Business Intelligence

Process analytics includes process querying, process analysis, matching and correlating process execution data and models [3]. According to Gartner, CIOs rank BI and analytics as number one technology priority for 2012 through 2016. Despite its central role in assessing the performance of organizations, process analytics, as of today, remains largely "behavior-unaware". A notable exception is the work done by the process mining community [1]. Mainstream BPM approaches tend to treat behavior models as static graphs rather than dynamic behavior generating artifacts. Mainstream BI approaches tend to be behavior agnostic. The topic of process querying, as defined in this paper, remains underdeveloped. A recent survey, refer to [4], demonstrates the lack of and the need for process querying methods grounded in behaviors encoded in models. Moreover, it is acknowledged that process analytics remains mostly limited in its ability to aggregate process performance indicators at the level of individual processes [3].

Process querying can be used to derive analytical insights about the performance of processes within organizations. We envision that this can be implemented via *process query procedures*, i.e., orchestrations of process queries. To bridge the gap between high-level business questions and process analytics,

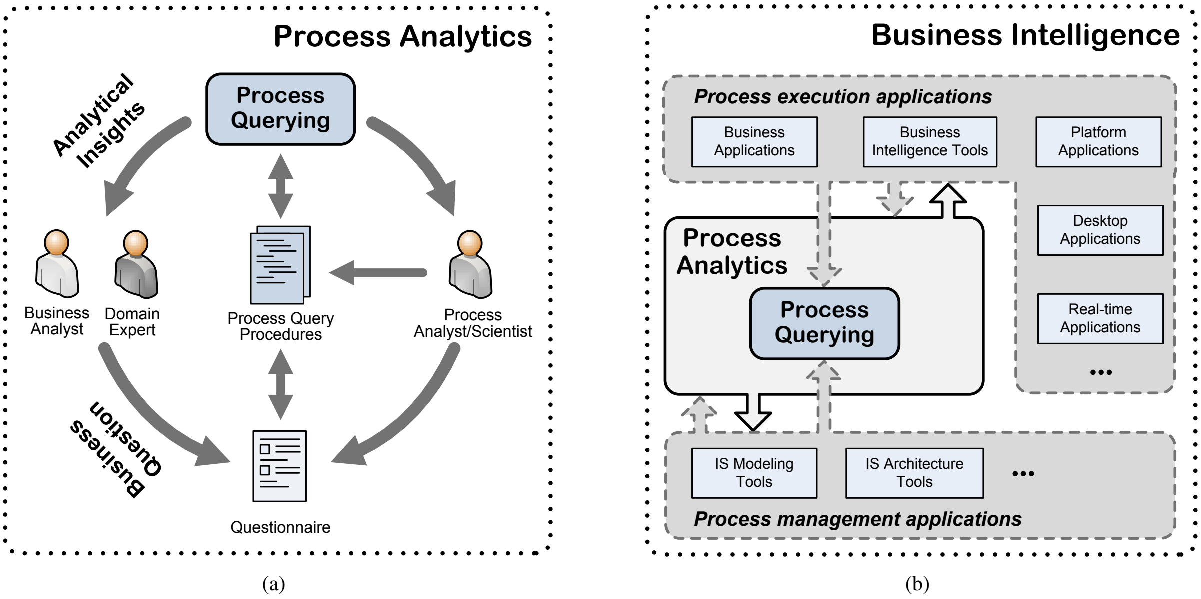

Figure 4. Positioning of process querying within (a) the Query-Based Process Analytics and (b) BI.

we propose to look into questionnaire-based approaches. By guiding *business analysts* and *domain experts* through preconfigured and intuitive (semi-automated) questionnaire instructions, one can attempt to translate business questions into low-level process query procedures that contribute towards answering the business question. In this light, a *business question* is a collection of inputs (i.e., existing process knowledge), outputs (i.e., derived process knowledge), and an analytical component. The analytical component of a business question is composed of manual and automated parts. Business questions with manual parts in analytical components seek expert knowledge, as every manual part is intended to be answered by a business analyst or domain expert. We suggest that automated parts of analytical components can be specified as process queries.

One can formalize a questionnaire as a state machine, where states encode available knowledge and transitions represent answers to manual parts or executions of automated parts of the corresponding analytical component. Hence, the user can start a questionnaire at one of its initial states, the one that encodes the current process knowledge, and proceed by executing (either manually or automatically) a predefined sequence of parts of the analytical component towards one of its accepting states, the one that represents the derived knowledge realizing the initial analytical need.

Figure 4(a) schematizes the envisaged high-level interactions between stakeholders and components when engaged into the query-based process analytics experience. Business analysts and domain experts can obtain an answer to a business question by completing the corresponding questionnaire. By working through a questionnaire, they configure process query procedures that realize automated parts of the analytical component of the question. The procedures enact process querying methods that deliver analytical insights back to the stakeholders who initiated the business question. The obtained insights are used to complete manual parts of the questionnaire. The results of the process query procedures influence the process of completing the questionnaire as they impact the acquired process knowledge. Once the questionnaire is completed, the accumulated analytical insights deliver an answer to the original business question. The role of a *process analyst*, or a *process scientist*, is to support process analytics experiences undergone by business analysts and domain experts. A process scientist is a person who has technical skills to design questionnaires and process query procedures, and to monitor executions of process querying methods. Questionnaires, as well as questionnaire patterns, that implement common process analytics scenarios, e.g., those that are available in the literature [3, 42], can be aggregated into an analytical knowledge-base. The users of the envisaged query-based process analytics approach will then be able to recompose (and again store for later reuse) existing and specify new questionnaires that realize emerging BI needs.

Figure 4(b) puts process querying and the proposed approach to process analytics into the broader perspective of BI. Business analysts refer to BI as a family of technologies that promise to deliver actionable information to support decision making. In our case, the angle is towards *process intelligence*, i.e., technologies that analyze the process-related information within process management and execution applications.

6. Process Querying: State of the Art

This section reports on a *structured literature review* of the existing research efforts that contribute to process querying. The review follows the guidelines proposed by Webster and Watson [46] and a more rigorous process suggested by vom Brocke et al [47]. The review process consists of three main phases: literature search and collection (Section 6.1), literature selection (Section 6.2), and literature analysis and evaluation (Section 6.3). To keep the process manageable with limited resources, instead of conducting a truly exhaustive literature review across various domains, we aim for a generous coverage of representative samples of the research work in the area of BPM on and closely related to the topic of process querying.

6.1. Literature Search and Collection

According to vom Brocke et al [47], before conducting a literature search, it is necessary to define a research scope and provide the conceptualization of the topic. As discussed in Sections 2–4, the topic of concern is process querying for managing collections of processes specified in the form of behavior models (e.g., event logs or process models). Based on the definition of the process querying problem, refer to Section 3, we identified four sets of keywords (see below) and used them to set the literature search criteria.

- **keywords-1**: *query*, *queries*, *querying*, *search*, *searches*, *searching*, *retrieve*, *retrieval*, *retrieving*, *analysis*, *analysing*, *analyzing*, *manage*, *managing*, *manipulate*, *manipulating*, *filter*, *filtering*
- **keywords-2**: *process*, *processes*, *workflow*, *workflows*, *model*, *models*
- **keywords-3**: *process model*, *process models*, *business process*, *business processes*, *event log*, *event logs*, *simulation model*, *simulation models*, *alignment*, *alignments*, *conformance checking*
- **keywords-4**: *repository*, *repositories*, *collection*, *collections*

For the literature search, we chose two popular scholarly databases, Scopus and Web of Science, which both are well-known sources of citations. For each database, we conducted two search iterations. Each iteration was performed *from scratch* against the whole database, instead of the search results from the previous iteration. We applied the following keyword-based search criteria:

- First iteration: ((*any word* in **keywords-1**) AND (*any word* in **keywords-2**)) in the paper's *Title* AND (*any word* in **keywords-3**) in (the paper's *Abstract* (if Scopus) OR the paper's *Topic* (if Web of Science)),
- Second iteration: (*any word* in **keywords-1**) AND (*any word* in **keywords-3**) AND (*any word* in **keywords-4**) in (the paper's *Abstract* (if Scopus) OR the paper's *Topic* (if Web of Science)).

In addition, we applied further filtering rules to literature search: Papers must be (i) written in English, (ii) published as a journal article, book, book chapter, proceedings paper, or as an article in press (Scopus only), (iii) published in the subject area of Computer Science in Scopus or in the Web of Science category of Computer Science Information Systems, and (iv) published between 2000 and 2017.

For each database, the search results of the two iterations were merged using the search engine of the database. Finally, the search results in the two databases were manually combined by removing the duplicates. This led to a collection of 2,647 papers. Table 1(a) summarizes the literature search statistics.

6.2. Literature Selection

The literature selection phase was performed in three sequential steps. In the first step, we identified and excluded irrelevant papers, as informed by their titles. The title of an excluded paper indicates that it is either beyond the scope of BPM or is not related to process querying. For example, we identified a large number of papers that study service computing (such as web search, service management, cloud, service-oriented model, etc.; 312 papers), simulation of systems or system models (but not of processes; 248 papers), data management (irrelevant to process querying; 231 papers), modeling in general (not on process or workflow modeling; 226 papers), alignment (of policies, systems, etc.; 152 papers), analysis of a process, workflow, log data (irrelevant to process querying; 167 papers), network analysis and networks of collaboration (123 papers), resource and organizational perspective (116 papers), software management (beyond the scope of BPM; 87 papers), enterprise system, model, and architecture (82 papers), security management (65 papers), and so on. As a result, $2,351$ out of $2,647$ collected papers were excluded based on their titles.

In the second step, we conducted selection based on the papers' topics. To this end, the abstract and content of a paper were studied to understand the topic of the paper. Our decision of selecting or excluding a paper was made according to the scope of this research as to study the relevant methods and techniques for querying business processes that are specified in the form of behavior models. Typical examples of excluded literature include: papers on scientific workflow as the topic has a different focus from business-oriented processes; papers on the application, rather than the definition, of supporting techniques (including process querying) for process analytics; papers on querying different aspects of a business process (e.g., its business content or performance) other than its control-flow; papers on querying a business process with a focus on the activity labels or the textual information of the process without considering the execution order of the activities in the process; etc. As a result, a total of 223 papers were removed based on their topics. In addition, we also discovered and removed 22 papers that are not written in English.

In the third step, we performed a paper quality screening. If a paper does not have a clear contribution or suffers from an unclear solution to the research question(s) raised in the paper, we do not consider it qualified for review and evaluation in the next phase. Another 12 papers were excluded due to quality issues. The selection phase led to a collection of 39 papers. Table 1(b) summarizes the selection statistics.

6.3. Literature Analysis and Evaluation

Prior to analysis of the 39 selected papers, we categorized them into four groups based on their topics.

- *Structural querying* focuses on the structural topology and characteristics of behavior models. Graph-based search techniques are mostly used in this group of research. Two typical examples are BP-QL for querying business processes specified in BPEL [48] and BPMN-Q for querying processes in BPMN [49–52]. Another research stream is on studying structural similarity between process models which provides useful underlying techniques for efficient structural querying [24, 53, 54]. Structural matching techniques propose ex-

(a)

Database	Iteration	# Hits Retrieved		
		By Iteration	By Database	Total
Scopus	first	1,586	2,386	2,647
	second	903		
Web of Science	first	486	773	
	second	336		

(b)

# Hits Retrieved	# Hits Excluded				# Hits Selected
	By Title	By Topic	*Non English**	By Quality	
2,647	2,351	223	22	12	39

** These papers were retrieved from Scopus only. They are written in a language other than English except for their title and abstract.*

Table 1. Statistics on (a) the literature search and (b) literature selection.

act matching between graphs or graph fragments and are applied in process querying [55, 56]. Two query languages [57, 58] that focus on pattern-based graph matching techniques are proposed. Other relevant work include: semantic querying of process structural characteristics [59], a querying technique used for BPMN process models annotated with cross-cutting concerns [60], a visual model query artifact [61], and a descriptive language to query and change the structural aspect of process models [62].

- *Behavioral querying* focuses on behaviors induced by activity models. Typical examples are querying techniques based on Petri nets [25, 63], YAWL nets [64], temporal logics such as LTL [65], and a process query language [66, 67] specifically designed to support behavioral querying. Another research stream studies behavioral similarity search [68–71] grounded in behavioral relations over activities and originates from the existing research on behavioral profile [72]. There is also a study on specifying and comparing processes using finite automata [73].
- *Process execution querying* addresses querying the execution traces of business processes at run-time (for monitoring purposes) or post execution. Both a process model and its event log are available inputs for the relevant research, and these are considered simulation models according to Definition 3.4. Typical examples are BP-Mon [74, 75], BP-Ex [76–78], and BPQL [79].
- *Event log querying* focuses on querying traces of business processes in the form of event logs only (i.e., no process model is available). Only a few relevant research outcomes were discovered [80–83]. Most of the approaches convert process logs into graphs and then apply FPSPARQL (an extension of SPARQL) [80] or graph-based search techniques [82, 83] to implement process querying. The authors of [81] use LTL to retrieve traces against a given business rule from the event log.

In addition, three survey papers on the topics of process model similarity search [84], behavioral similarity metrics and evaluation [85], and process querying techniques [4] were selected. In [4], the authors focus on comparing syntactic-based querying and semantic-based querying to emphasize the importance of considering the process semantics (i.e., behavior) when querying process repositories.

Before moving to a detailed evaluation of the selected literature, we defined a set of evaluation metrics as informed by the PQF, refer to Section 4. To validate the framework's design, we were interested in understanding the selected literature in the following four aspects.

- *Process Repository Design* aspect:
  - What types of *behavior models* are supported by process repositories?
  - What *modeling languages* are used to specify behavior models?
  - What is the *level of formalism* of behavior models in process repositories? For example, a process model may be specified in a formal language such as Petri net, a semi-formal language such as BPMN (which has execution semantics), or an informal language like value chains; refer to Section 3 for more details on formal, semi-formal, and informal behavior models.
- *Process Query Design* aspect:
  - What *query intents* are supported? The *Read* intent is further subdivided into two intents of 'retrieve' and 'project', where 'retrieve' is to know whether or not a model satisfies a query and 'project' is to obtain some details about a model (e.g., a fragment of the model) that satisfies a query.
  - What underlying techniques are applied to support the querying (*querying technique* for short)?
  - What is the level of formalism with respect to the semantics of querying method (*querying semantics* for short)? In other words, up to what level of formalism does the querying method treat behavior models in process repositories. For example, the process repository may be composed of formal models but the proposed querying method may be treating them as informal models.
  - Is there a particular research artifact in the form of a *query language*?
- *Prepare & Execute* aspect looks into which components of the "Prepare" part and the "Execute" part of the framework are studied in the literature.
- *Interpret* aspect studies which components of the "Interpret" part of the framework for explaining process querying results are addressed in the state of the art querying methods.

Next, we analyzed the main research efforts from the selected literature in process querying using the above evaluation metrics. The papers are organized into the four topic groups that were mentioned earlier, and one key paper is included for each research item. Note that the three survey papers are not included in the literature evaluation. Detailed evaluation results of these key papers are presented in Table 2.

The literature analysis results in Table 2 justify the design of the framework proposed in Section 4, as most of the proposed components and features of the design are addressed in the literature. None of the existing methods for process querying addresses all the components of the framework, even for a

Literature			Process Repository Design																Process Query Design																													Prepare & Execute			Intepret		
			behavior model			modeling language										level of formalism			query intent			querying technique									querying semantic			query language																			
Topic Group	(Short) Title	key reference	event log	simulation model	process model	BPMN	BPAL (a BPMN variant)	BPEL	EPC	finite state automaton (FSA)	Petri net	YAWL net	process graph	direct acyclic graph	process metamodel	formal	semi-formal	informal	Read (retrieve)	Read (project)	Update	graph matching	graph similarity	graph-based search	FSA-based	Petri net-based	temporal logic	event correlation	top-k selection	activity label similarity	formal - partial order	semi-formal - partial order	informal	BP-QL	BP-Mon	BP-Ex	BPMN-Q	BPMN VQL	DMQL	GMQL	VMQL	Descriptive PQL	BQL	APQL / PQL	FPSPARQL	QuBPAL	BPQL	indexing	filtering	optimizing (exec. plan)	projecting	translating	visualizing
Structural querying	Querying business processes with BP-QL	[48]			√			√									√			√		√										√		√																			√
	BPMN-Q: A language to querying business processes	[49]			√	√											√		√	√		√										√					√											±					√
	Semantic querying of business process models	[50]			√	√											√		√	√		√								√		√					√																√
	Querying graph-based repositories of process models	[51]			√	√											√		√	√		√										√					√											±	±	√			√
	Efficient processing of BPMN-Q queries	[52]			√	√											√		√	√		√										√					√											√	√	√			√
	Query structural information of BPEL processes	[55]			√								√					√	√				√									√																					√
	Process Matching: A structural approach for process search	[56]			√								√					√	√			√										√																±					√
	FNet: An index for advanced business process querying	[24]			√								√				√		√			√										√																√		√			√
	Ontology-based querying of composite services by processes	[59]			√		√											√	√					√								√														√						√	
	Crosscutting concern documentation by BPMN VQL	[60]			√	√												√	√					√								√						√															√
	DMQL: The diagramed model query language	[57]			√				√									√	√	√		√										√							√														√
	GMQL: The generic model query language	[58]			√								√				√		√	√		√										√								√													√
	Querying business process models with VMQL	[61]			√	√												√	√	√		√										√									√												√
	PQL: A descriptive language to query, abstract & change process models	[62]			√								√					√	√		√	√										√										√											√
Behavioral querying	Indexing business processes via annotated finite state automata	[73]			√					√							√		√						√							√																√				√	
	Compliance checking using BPMN-Q & temporal logic	[65]			√	√											√		√	√		√				√	√				√						√											√					√
	Querying business process models based on semantics	[63]			√						√					√			√							√					√												√					±				√	
	Efficient querying of large process model repositories	[64]			√							√				√			√							√				√	√												√					√				√	
	APQL: A process-model query language	[66]			√	√		√			√	√				√	√		√																									√								√	
	Indexing and efficient instance-based retrieval of process models	[25]			√						√					√			√							√					√													√				√		√		√	
	Querying process models by behavior inclusion	[71]			√														√							√					√																	√	√			√	
Process execution querying	Monitoring business processes with queries	[74]		√				√						√			√		√			√										√			√														√			√	√
	Type inference and checking for queries on execution traces	[76]		√				√						√			√		√			√										√				√														√			√
	Models and query languages for probabilistic processes	[78]		√				√						√			√		√	√		√							√			√				√														√	√	√	√
	Process query language to make workflow processes more flexible	[79]		√											√			√	√	√													√														√					√	
Event log querying	A query language for analyzing business processes execution	[80]	√													√			√	√				√				√				√													√								√
	Log-based understanding of processes via temporal logic query checking	[81]	√													√			√								√				√																					√	
	Log-based process fragment querying to support process design	[82]	√													√			√	√				√								√																					√
	A framework supporting querying analysis of process logs	[83]	√													√			√					√								√																				√	

NOTE: The sign of √ indicates the evaluation item is supported. The sign of ± means there is some intermediate or indirect support. E.g., [51] applies the index-and-filtering functionalities provided by the underlying RDBMS deployed in the research instead of developing its own mechanism, hence it is evaluated ± for both items of indexing and filtering. Then, in [52] the authors progressed the work by proposing their own mechanisms for indexing and filtering and therefore the work is evaluated √ for both items.

Table 2. Evaluation results of the key papers on process querying in the BPM field.

subset of behavior model types and/or different levels of formalism. The insights gained from the reported evaluation were used to inform the design of the framework. In particular, the "Interpret" part was introduced after the respective functionality was identified in the selected literature.

The identified in this literature review research gaps include: (i) absence of querying methods over correlation models, e.g., alignments [36], (ii) absence of unified querying methods over several types of behavior models, (iii) absence of unified querying methods over all levels of formalism of behavior models, e.g., informal, semi-formal, and formal models, (iv) absence of works on caching in the context of process querying, (v) lack of techniques for indexing process repositories and optimization of process queries, (vii) lack of various means for explaining process querying results (as most of the methods rely only on means of translating and visualizing querying results), and (viii) lack of works on components that can be used to instantiate the proposed framework for querying over formal process models with querying semantics grounded in behavior of the process models. We believe that the proposed framework for process querying methods and the insights gained in the conducted literature review will help to steer as of today somewhat uncoordinated efforts in research on process querying.

7. Conclusion

This paper proposes a framework for developing process querying methods, i.e., (automated) techniques for managing process repositories. The active components of the framework specify generic functionalities that can be configured and specialized to address a particular process querying problem. The framework is grounded in various use cases taken from the BPM field and the reported literature review. The use cases motivate the framework and guide its design. The literature review justifies the design and reveals research gaps. As of today, comprehensive process querying methods grounded in behaviors captured by models of dynamic systems are missing. The introduction of these methods will enable the so far unmatched experience in process analytics that will lead to the next generation of smart BI technologies.

Acknowledgements. This research is partly supported by the Australian Research Council Discovery Project DP150103356.

References

[1] W. M. P. van der Aalst, Process Mining—Data Science in Action, 2nd ed., Springer, 2016.

[2] M. zur Muehlen, R. Shapiro, Business process analytics, in: Handbook on Business Process Management 2, International Handbooks on Information Systems, 2nd ed., Springer, 2015, pp. 243–263.

[3] S. Beheshti, B. Benatallah, S. Sakr, D. Grigori, H. R. Motahari-Nezhad, M. C. Barukh, A. Gater, S. H. Ryu, Process Analytics—Concepts and Techniques for Querying and Analyzing Process Data, Springer, 2016.

[4] J. Wang, T. Jin, R. K. Wong, L. Wen, Querying business process model repositories—a survey of current approaches and issues, World Wide Web 17 (2014) 427–454.

[5] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manage. Inform. Syst. 24 (2008) 45–77.

[6] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004) 75–105.

[7] W. M. P. van der Aalst, Business process management: A comprehensive survey, ISRN Software Engineering 2013 (2013).

[8] J. L. Whitten, V. M. Barlow, L. Bentley, Systems Analysis and Design Methods, 3rd ed., McGraw-Hill Professional, 1997.

[9] A. Osterwalder, Y. Pigneur, Designing business models and similar strategic objects: The contribution of IS, J. Assoc. Inf. Syst. 14 (2013).

[10] J. A. Zachman, A framework for information systems architecture, IBM Syst. J. 38 (1999) 454–470.

[11] V. Haren, TOGAF Version 9.1, 10th ed., Van Haren Publishing, 2011.

[12] A. P. Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, A. Wood, Processes, roles, and events: UML concepts for enterprise architecture, in: UML, vol. 1939 of *LNCS*, Springer, 2000, pp. 62–77.

[13] H. Jonkers, M. M. Lankhorst, R. van Buuren, S. Hoppenbrouwers, M. M. Bonsangue, L. W. N. van der Torre, Concepts for modeling enterprise architectures, Int. J. Coop. Inf. Syst. 13 (2004) 257–287.

[14] M. Weidlich, A. P. Barros, J. Mendling, M. Weske, Vertical alignment of process models—how can we get there?, in: BPMDS, vol. 29 of *LNBIP*, Springer, 2009, pp. 71–84.

[15] A. Polyvyanyy, M. Weidlich, M. Weske, Isotactics as a foundation for alignment and abstraction of behavioral models, in: BPM, vol. 7481 of *LNCS*, Springer, 2012, pp. 335–351.

[16] T. Schneider, SAP Business ByDesign Studio—Application Development, 2011. SAP Press.

[17] X. Lu, M. Nagelkerke, D. van de Wiel, D. Fahland, Discovering interacting artifacts from ERP systems, IEEE Trans. Serv. Comput. 8 (2015).

[18] A. Koschmider, M. Fellmann, A. Schoknecht, A. Oberweis, Analysis of process model reuse: Where are we now, where should we go from here?, Decis. Support Syst. 66 (2014) 9–19.

[19] B. F. van Dongen, R. M. Dijkman, J. Mendling, Measuring similarity between business process models, in: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE, Springer, 2013, pp. 405–419.

[20] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers, Merging event-driven process chains, in: OTM Conferences, vol. 5331 of *LNCS*, Springer, 2008, pp. 418–426.

[21] I. Trummer, C. Koch, Multi-objective parametric query optimization, PVLDB 8 (2014) 221–232.

[22] D. Fahland, W. M. P. van der Aalst, Model repair—aligning process models to reality, Inf. Syst. 47 (2015) 220–243.

[23] A. Polyvyanyy, W. M. P. van der Aalst, A. H. M. ter Hofstede, M. T. Wynn, Impact-driven process model repair, ACM Trans. Softw. Eng. Methodol. 25 (2017) 1–60.

[24] Z. Yan, R. Dijkman, P. Grefen, FNet: An index for advanced business process querying, in: BPM, vol. 7481 of *LNCS*, Springer, 2012, pp. 246–261.

[25] A. Polyvyanyy, M. La Rosa, A. H. M. ter Hofstede, Indexing and efficient instance-based retrieval of process models using untanglings, in: CAiSE, vol. 8484 of *LNCS*, Springer, 2014, pp. 439–456.

[26] C. A. Petri, Nicht-sequentielle Prozesse, Arbeitsberichte des IMMD 8, Universität Erlangen Nürnberg, 1976.

[27] M. Nielsen, G. D. Plotkin, G. Winskel, Petri nets, event structures and domains, Part I, Theor. Comput. Sci. 13 (1981).

[28] U. Goltz, W. Reisig, The non-sequential behavior of Petri nets, Inf. and Cont. 57 (1983).

[29] O. I. Lindland, G. Sindre, A. Sølvberg, Understanding quality in conceptual modeling, IEEE Softw. 11 (1994) 42–49.

[30] T. H. Cormen, Algorithms Unlocked, MIT Press, 2013.

[31] J. Recker, Evidence-based business process management: Using digital opportunities to drive organizational innovation, in: BPM—Driving Innovation in a Digital World, Springer, 2015, pp. 129–143.

[32] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.

[33] J. Banks, J. S. C. II, B. L. Nelson, D. M. Nicol, Discrete-Event System Simulation, 5th ed., Pearson Education, 2010.

[34] W. M. P. van der Aalst, C. Stahl, Modeling Business Processes—A Petri Net-Oriented Approach, MIT Press, 2011.

[35] W. Reisig, Understanding Petri Nets—Modeling Techniques, Analysis Methods, Case Studies, Springer, 2013.

[36] W. M. P. van der Aalst, A. Adriansyah, B. F. van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdiscip. Rev.-Data Mining Knowl. Discov. 2 (2012)

182–192.

[37] J. Mendling, M. Strembeck, J. Recker, Factors of process model comprehension—findings from a series of experiments, Decis. Support Syst. 53 (2012) 195–206.

[38] J. Mendling, H. A. Reijers, W. M. P. van der Aalst, Seven process modeling guidelines (7PMG), Inf. Softw. Technol. 52 (2010).

[39] M. Sipser, Introduction to the Theory of Computation, 3rd ed., Cengage Learning, 2012.

[40] E. Kindler, W. M. P. van der Aalst, Liveness, fairness, and recurrence in Petri nets, Inf. Process. Lett. 70 (1999).

[41] K. R. Apt, N. Francez, S. Katz, Appraising fairness in languages for distributed programming, Distrib. Comput. 2 (1988).

[42] M. Reichert, B. Weber, Enabling Flexibility in Process-Aware Information Systems, Springer, 2012, pp. 297–317.

[43] J. Esparza, M. Nielsen, Decidability issues for Petri nets—a survey, Bulletin of the EATCS 52 (1994) 244–262.

[44] R. W. White, R. A. Roth, Exploratory Search: Beyond the Query-Response Paradigm, Morgan & Claypool Publishers, 2009.

[45] M. Dwyer, G. Avrunin, J. Corbett, Patterns in property specifications for finite-state verification, in: ICSE, ACM, 1999, pp. 411–420.

[46] J. Webster, R. T. Watson, Analyzing the past to prepare for the future: Writing a literature review, MIS Q. 26 (2002) xiii–xxiii.

[47] J. Vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, A. Cleven, et al., Reconstructing the giant: On the importance of rigour in documenting the literature search process., in: ECIS, 2009, pp. 2206–2217.

[48] C. Beeri, A. Eyal, S. Kamenkovich, T. Milo, Querying business processes with BP-QL, Inf. Syst. 33 (2008) 477–507.

[49] A. Awad, BPMN-Q: A language to query business processes, in: EMISA, vol. P-119 of *LNI*, GI, 2007, pp. 115–128.

[50] A. Awad, A. Polyvyanyy, M. Weske, Semantic querying of business process models, in: EDOC, IEEE Comp. Society, 2008, pp. 85–94.

[51] A. Awad, S. Sakr, Querying graph-based repositories of business process models, in: DASFAA Workshops, vol. 6193 of *LNCS*, Springer, 2010, pp. 33–44.

[52] A. Awad, S. Sakr, On efficient processing of BPMN-Q queries, Comput. Ind. 63 (2012) 867–881.

[53] R. Dijkman, M. Dumas, L. García-Bañuelos, Graph matching algorithms for business process model similarity search, in: BPM, vol. 5701 of *LNCS*, Springer, 2009, pp. 48–63.

[54] Z. Yan, R. Dijkman, P. Grefen, Fast business process similarity search, Distrib. Parallel Databases 30 (2012) 105–144.

[55] Z. Ma, W. Lu, F. Leymann, Query structural information of BPEL processes, in: ICIW, IEEE Computer Society, 2009, pp. 401–406.

[56] J. Zhu, H. K. Pung, Process matching: A structural approach for business process search, in: ComputationWorld, IEEE Computer Society, 2009, pp. 227–232.

[57] P. Delfmann, D. Breuker, M. Matzner, J. Becker, Supporting information systems analysis through conceptual model query—the diagramed model query language (DMQL), Communications of the Association for Information Systems 37 (2015).

[58] P. Delfmann, M. Steinhorst, H.-A. Dietrich, J. Becker, The generic model query language GMQL—conceptual specification, implementation, and runtime evaluation, Inf. Syst. 47 (2015) 129–177.

[59] F. Smith, M. Missikoff, M. Proietti, Ontology-based querying of composite services, in: BSME, Springer, 2010, pp. 159–180.

[60] C. Di Francescomarino, P. Tonella, Crosscutting concern documentation by visual query of business processes, in: BPM Workshops, vol. 17 of *LNBIP*, Springer, 2008, pp. 18–31.

[61] H. Störrle, V. Acretoaie, Querying business process models with VMQL, in: BMFA, 4, ACM, 2013, pp. 1–10.

[62] K. Kammerer, J. Kolb, M. Reichert, PQL—a descriptive language for querying, abstracting and changing process models, in: BPMDS, vol. 214 of *LNBIP*, Springer, 2015, pp. 135–150.

[63] T. Jin, J. Wang, L. Wen, Querying business process models based on semantics, in: DASFAA, Springer, 2011, pp. 164–178.

[64] T. Jin, J. Wang, M. La Rosa, A. ter Hofstede, L. Wen, Efficient querying of large process model repositories, Comput. Ind. 64 (2013).

[65] A. Awad, G. Decker, M. Weske, Efficient compliance checking using BPMN-Q and temporal logic, in: BPM, vol. 5240 of *LNCS*, Springer, 2008, pp. 326–341.

[66] A. H. M. ter Hofstede, C. Ouyang, M. La Rosa, L. Song, J. Wang, A. Polyvyanyy, APQL: A process-model query language, in: AP-BPM, vol. 159 of *LNBIP*, Springer, 2013, pp. 23–38.

[67] A. Polyvyanyy, L. Corno, R. Conforti, S. Raboczi, M. La Rosa, G. Fortino, Process querying in Apromore, in: BPM Demos, vol. 1418 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 105–109.

[68] M. Kunze, M. Weske, Metric trees for efficient similarity search in large process model repositories, in: BPM, vol. 66 of *LNBIP*, Springer, 2010, pp. 535–546.

[69] M. Kunze, M. Weidlich, M. Weske, Behavioral similarity—a proper metric, in: BPM, vol. 6896 of *LNCS*, Springer, 2011, pp. 166–181.

[70] M. Guentert, M. Kunze, M. Weske, Evaluation measures for similarity search results in process model repositories, in: ER, vol. 7532 of *LNCS*, Springer, 2012, pp. 214–227.

[71] M. Kunze, M. Weidlich, M. Weske, Querying process models by behavior inclusion, Softw. Syst. Model. 14 (2015).

[72] M. Weidlich, H. Ziekow, J. Mendling, Optimising complex event queries over business processes using behavioural profiles, in: BPM Workshops, vol. 66 of *LNBIP*, Springer, 2010, pp. 743–754.

[73] B. Mahleko, A. Wombacher, Indexing business processes based on annotated finite state automata, in: ICWS, IEEE, 2006, pp. 303–311.

[74] C. Beeri, A. Eyal, T. Milo, A. Pilberg, Monitoring business processes with queries, in: VLDB, ACM, 2007, pp. 603–614.

[75] C. Beeri, A. Eyal, T. Milo, A. Pilberg, BP-Mon: Query-based monitoring of BPEL business processes, Sigmod Rec. 37 (2008).

[76] D. Deutch, T. Milo, Type inference and type checking for queries on execution traces, PVLDB 1 (2008) 352–363.

[77] E. Balan, T. Milo, T. Sterenzy, BP-Ex: A uniform query engine for business process execution traces, in: EDBT, ACM, 2010, pp. 713–716.

[78] D. Deutch, T. Milo, On models and query languages for probabilistic processes, Sigmod Rec. 39 (2010) 27–38.

[79] M. Momotko, K. Subieta, Process query language: A way to make workflow processes more flexible, in: ADBIS, vol. 3255 of *LNCS*, Springer, 2004, pp. 306–321.

[80] S. Beheshti, B. Benatallah, H. R. M. Nezhad, S. Sakr, A query language for analyzing business processes execution, in: BPM, vol. 6896 of *LNCS*, Springer, 2011, pp. 281–297.

[81] M. Räim, C. Di Ciccio, F. M. Maggi, M. Mecella, J. Mendling, Log-based understanding of business processes through temporal logic query checking, in: OTM Conferences, vol. 8841 of *LNCS*, Springer, 2014, pp. 75–92.

[82] K. Yongsiriwit, N. N. Chan, W. Gaaloul, Log-based process fragment querying to support process design, in: HICSS, IEEE Computer Society, 2015, pp. 4109–4119.

[83] B. Fazzinga, S. Flesca, F. Furfaro, E. Masciari, L. Pontieri, C. Pulice, A framework supporting the analysis of process logs stored in either relational or NoSQL DBMSs, in: ISMIS, vol. 9384 of *LNCS*, Springer, 2015, pp. 52–58.

[84] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: Metrics and evaluation, Inf. Syst. 36 (2011) 498–516.

[85] M. Kunze, M. Weske, Methods for evaluating process model search, in: BPM Workshops, vol. 171 of *LNBIP*, Springer, 2013, pp. 379–391.